Indian Statistical Institute
Kolkata

M.Tech. (Computer Science) Dissertation

Search in Transliterated Domain for Marathi

A dissertation submitted in partial fulfillment of the requirements
for the award of Master of Technology
in
Computer Science

Author:
Dnyaneshwar Shivshankar Patil
Roll No: CS 1311

Supervisor:
Dr. Mandar Mitra
Computer Vision and Pattern
Recognition Unit

CERTIFICATE

M.TECH(CS) DISSERTATION THESIS COMPLETION CERTIFICATE

Student : Dnyaneshwar Shivshankar Patil (CS 1311)

Topic : Search in Transliterated Domain for Marathi

Supervisor : Dr. Mandar Mitra

This is to certify that the thesis titled **Search in Transliterated Domain for Marathi** submitted by **Dnyaneshwar Shivshankar Patil** in partial fulfillment for the award of the degree of Master of Technology is a bonafide record of work carried out by him under our supervision. The thesis has fulfilled all the requirements as per the regulations of this Institute and, in our opinion, has reached the standard needed for submission. The results embodied in this thesis have not been submitted to any other university for the award of any degree or diploma.

Dr. Mandar Mitra
Computer Vision and Pattern Recognition Unit
Indian Statistical Institute

Date : 13^{th} *July*, 2015

Acknowledgement

First of all, I wish to express my sincere gratitudes to **Dr. Mandar Mitra** for suggesting me this topic in the first place and helping me with the challenges that I have faced. Without his guidance and immense support, this project would not have been possible.

I am very grateful to all my classmates especially Kunal Ray, Chirag Gupta, Ravindra Kumar and Ansuman Dash for helping me in technical matters and discussing different ideas.

Abstract

A large number of languages, including Arabic, Russian, and most of the South and South East Asian languages, are written using indigenous scripts. However, often the Webster and the user generated content in these languages are written using Roman script for various reasons. A challenge that search engines face while processing transliterated queries and documents is that of extensive spelling variation.

In this report, we handle the word language identification problem for Marathi language which is written in Roman script and also has English language words. We have considered a method based on character level n-grams, to address this problem. Our method gives around 98% mean accuracy using 10-fold cross validation. In addition to word language identification, we also handle transliteration of Marathi language words from Roman script to Devanagari script. Our method provides around 95% character level unigram precision.

We have also implemented an ad-hoc retrieval for Marathi news data based on transliterated queries. We compared the results obtained with the original Devanagari script queries and found that the performance of ad-hoc retrieval system did not deteriorate significantly.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction to Transliteration

A large number of languages, including Arabic, Russian, and most of the South and South East Asian languages, are written using indigenous scripts. However, often the Webster and the user generated content (such as tweets and blogs) in these languages are written using Roman script due to various socio-cultural and technological reasons such as lack of standard keyboards, a large number of scripts and familiarity with English & QWERTY keyboards[1].

1.1.1 What is Transliteration?

Transliteration refers to the process of writing the text of one language using the script of another language whereby the sound of the text is preserved as far as possible [1]. Depending upon the direction of Transliteration, it can be divided into two types[6].

1. Forward Transliteration:- Forward Transliteration is the process by which a word in a native language (in our case, Marathi) with a native script (in our case, Devanagari) is represented in a non-native script (in our case, Roman). For Example, *Dnyaneshwar* is generated in Roman script by forward transliterating the word ज्ञानेश्वर in Marathi written in Devanagari script

2. Backward Transliteration:- Backward Transliteration is the process of obtaining a word in native script from the transliterated word, for example, getting back ज्ञानेश्वर from *Dnyaneshwar*

Transliteration in Roman script is used abundantly on web, not only for writing documents but for queries as well.

1.2 Mixed Script Information Retrieval (MSIR) [5]

Webster and user generated content (tweets, Facebook posts, blogs etc.) in native languages (in our case Marathi), may be written either in Roman or in the native script . This content creates a monolingual

[1] http://research.microsoft.com/en-us/events/fire13_st_on_transliteratedsearch/FIRE14ST.aspx

or multi-lingual space which is in more than one scripts. Such a space is called *Mixed-Script Space*.

Information retrieval done on a mixed-script space is called *Mixed-Script Information Retrieval (MSIR)*

1.3 Motivation[5]

In a study performed on 13 billion queries on Bing search engine [2] issued from India, 6% of the distinct queries had one or more Hindi words transliterated into Roman script, of which 28% queries were Named Entities (NE) and 27% belonged to the entertainment domain (e.g.. movie name, song lyrics, dialogues etc.). Hindi songs were one of the most searched items in India[5]. This motivated a study on MSIR for Hindi Song Lyrics. To the best of my knowledge, no such study is available for Marathi language. However, we believe similar observations can be made for Marathi as well.

In the year 2013, a shared transliteration task[3] was introduced at *Forum for Information Retrieval and Evaluation (FIRE)*[4] track by Microsoft Research, India. It was for an ad-hoc task that involved searching for Hindi song lyrics.

Before the ad-hoc search task, 2 subtasks have to be completed on a given query as part of preprocessing.

1. Query Word Labeling :- To label each word in a query with its language. The list of possible languages are known to us in advance.

2. Backward transliteratation :- Backward transliteration of words which are labelled as non-English to native script (in our case Devanagari)

Following example will make the purpose of above tasks more clear. English words will be labelled by 'E' and Marathi words will be labelled by 'M'.

Input	**Query word labeling**	**Backward transliteration**
paalak paneer recipe	paalak\M paneer\M recipe\E	पालक पनीर recipe
maajhe jivangane	maajhe\M jivangane\M	माझे जीवनगाणे
thank you very much	thank\E you\E very\E much\E	thank you very much

Table 1.1: Examples : Query word labeling and Backward transliteration

Data in many Indian languages was available for these 2 subtasks. However, till last year, no Marathi data was present. This motivated us to create a dataset for these 2 subtasks for Marathi.

[2] http://www.bing.com/
[3] http://research.microsoft.com/en-us/events/fire13_st_on_transliteratedsearch/default.aspx
[4] http://fire.irsi.res.in/

1.4 Challenges

- To handle spelling variations :- Since there is no standard way to do a forward transliteration of native word into non-native script, transliterated documents and queries involve extensive spelling variations[5]. For example, the Marathi word धन्यवाद (*thanks* in Marathi and many other Indian languages), can be written in Roman script as *dhanywad, dhanyavad, dhanyavaad, dhanyabad, danyavad* and so on. These spelling variations create a problem while matching a query, which is in Devanagari or Roman script, with documents which are also in either of these two scripts.
- Script specification :- In both the transliterations i.e. *forward* and *backward*, the source and target scripts are different. The backward transliteration system needs to tackle this hurdle. [6]

1.5 Problem Statement

To create a dataset and to build an Information Retrieval (IR) system for Marathi,which will involve following subtasks:

1. Query word language identification and labeling;

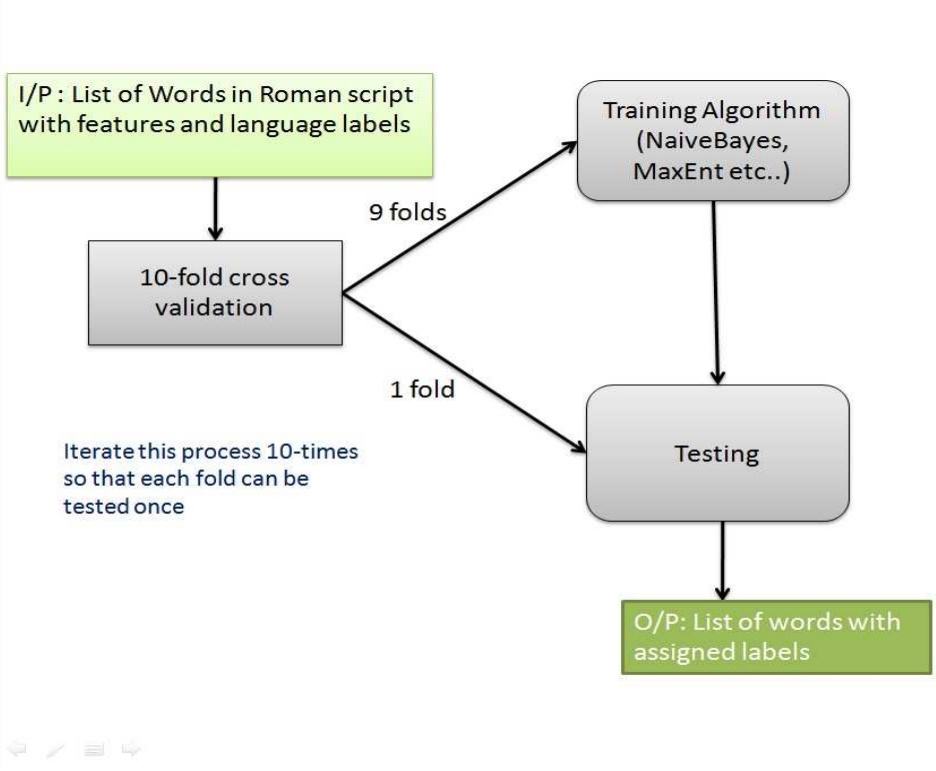

Figure 1.1: Block Diagram for *Query Word Labeling*

2. Backward transliteration of words in query labelled as Marathi

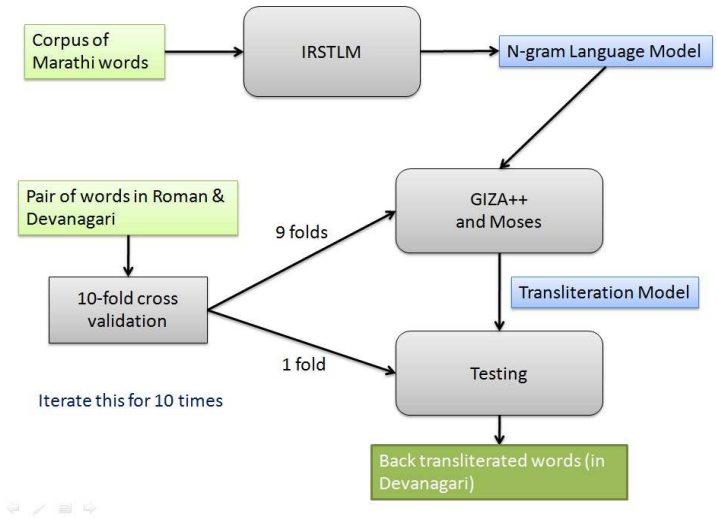

Figure 1.2: Block Diagram for *Backward Transliteration*

3. Ad Hoc retrieval on Marathi data

We perform this subtask on News data available in FIRE 2010 instead Marathi songs. In the block diagram below, for *Query word labeling & Backward Transliteration*, refer to figure 1.1 & 1.2

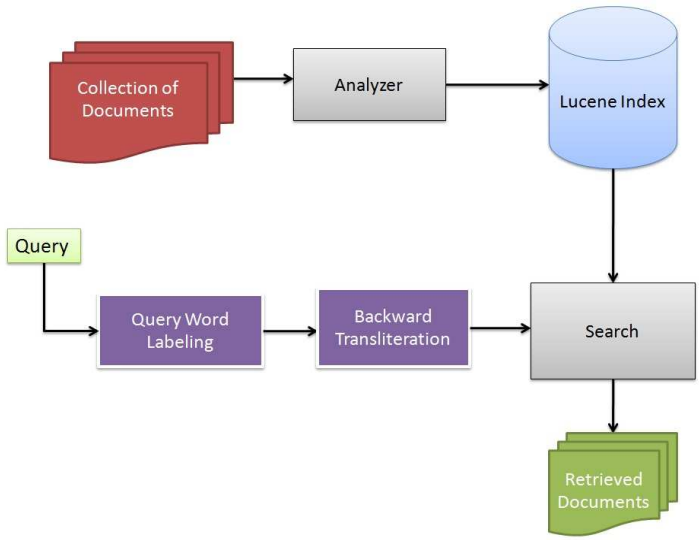

Figure 1.3: Block Diagram for *Ad Hoc Retrieval*

Chapter 2

Related Work

2.1 Dataset creation

To build a transliteration system for Indian Languages, a rigorous dataset creation process was described in [1]. In this paper, three Indian languages (Hindi, Bengali, Telugu) were chosen to collect data. The transliterated data was obtained from users using two settings:

1. Dictation or Controlled setting :- In this setting, the text that a user enters is decided *apriori* and thus, the user does not have control over it. This is done to ensure the language coverage (covering as many vowel and consonant combinations as possible). Each user was provided an audio recording of some sentences in Indian languages and asked to transcribe the sentences in Roman script.

2. Uncontrolled setting :- In this setting, the user can construct his/her own sentences. In this setting, two modes, namely *Scenario* and *Chat*, were used. *Scenario* was used to model blogs or email, and *Chat* was used to collect chatting data. In *Scenario* mode, the user has time to think, write and edit; however, there is time pressure in *Chat* mode.

2.2 Query Word Labeling

King & Abney, in their 2013 paper [7], have described a word language labeling method, which is weakly supervised, for mixed-script documents. This method uses a combination of character level unigram, bigram, trigram, 4-gram, 5-gram and full words as feature set. This labeling method was used by MSRI[4] for labeling query words in 3 Indian languages, namely *Hindi, Bengali and Gujarati*. The system submitted by this team showed the best results. MSRI used different classifiers provided by MALLET [11] such as NaiveBayes, MaxEnt, Decision Tree etc.
In FIRE 2013[1], other teams also submitted systems based on various algorithms such as simple lookup[2], character level n-gram with SVM classifier[3] etc.

[1] http://www.isical.ac.in/~fire/2013/working-notes.html
[2] http://www.isical.ac.in/~fire/wn/STTS/2013-translit_search-prabhakar-ism_dhanbad.pdf
[3] http://www.isical.ac.in/~fire/wn/STTS/2013-translit_search-gupta-tu_valencia.pdf

In FIRE 2014[4], many teams submitted their systems which used methods such as graph based approach (LIGA)[5], SVM classifier[6], Naive Bayes classifier in conjunction with various token features[7].

2.3 Backward Transliteration

In a machine transliteration survey done by *Karimi, Scholer & Turpin*[6], they have reviewed various methods and approaches in machine transliteration literature based on effectiveness, resource and algorithms used. According to the survey, machine transliteration emerged as part of machine translation to translate proper names and technical terms. As per the survey, there have been various methods which were used to transliterate names, phonebook entries, words from literature etc.
David Matthews, in his Master's thesis, used Moses [9] for backward transliteration and it outperformed a baseline model (which was roughly based on average length ratios) in both directions for English-Chinese and Arabic-English.

In FIRE 2013, MSRI team had used a hash function based Name Search tool which would convert words from Roman Script to Devanagari script. After this, with the help of Indic mapping, they converted the characters to their own languages if the target language was Gujarati or Bengali[4]. The approaches include syllabification approach[8], trigram model, joint source channel model[9] etc.
In FIRE 2014, many approaches such as Google Transliteration API [6], rule based approach[10] etc. were used for backward transliteration.

[4] http://www.isical.ac.in/~fire/2014/working-notes.html
[5] http://www.isical.ac.in/~fire/working-notes/2014/MSR/FIRE2014_DA-IR.pdf
[6] http://www.isical.ac.in/~fire/working-notes/2014/MSR/FIRE2014_BITS-Lipyantran.pdf
[7] http://www.isical.ac.in/~fire/working-notes/2014/MSR/FIRE2014_I1.pdf
[8] http://www.isical.ac.in/~fire/wn/STTS/2013-translit_search-joshi-guj_univ.pdf
[9] http://www.isical.ac.in/~fire/wn/STTS/2013-translit_search-pakray-ntnu-norway.pdf
[10] http://www.isical.ac.in/~fire/working-notes/2014/MSR/FIRE2014_BMS-Brainz.pdf

Chapter 3

Approach

3.1 Query Word Labeling

After studying different methods which were used in FIRE 2013 and 14 as well as some other research papers, a method involving character level n-gram was chosen. This method was briefly described in Section 2.2. We divide each word into character level n-grams where n ranges from 1 to 5. With these n-grams and the complete word as a feature, we train classifiers such as Naive Bayes, Maximum Entropy, Decision Tree and Winnow. *King & Abney* in their paper used a weakly supervised method. However, our training data is labeled, therefore we will run supervised learning algorithms. We will use 10-fold cross validation method to test the accuracy.

Also, we will use dictionary based approach after learning algorithm gives output, to check whether a particular word belongs to English or not. The data is very skewed in favour of Marathi words and thus we do not have enough data for learning algorithm to learn about the English words. We check whether a Marathi classified word is really an English word or not with help of dictionary. If there is conflict that the spelling in present in both scripts (e.g. man), then we keep the label of that word as given by learning algorithm.

3.2 Backward Transliteration

For backward transliteration, the approaches mentioned in FIRE 2013 and 14 as well as a thesis by *David Matthews* [10] were studied. It was decided to design a system which will do transliteration based on Moses toolkit. This involves three steps[1]:

1. Creating a Language model for Marathi
2. Building a transliteration model for transliterating word from Roman to Devanagari
3. Given a word in Roman script, transliterate it to Devanagari using the transliteration model created in the previous step

[1] http://www.cs.columbia.edu/~mcollins/courses/nlp2011/notes/ibm12.pdf

In the rest of this section, we will describe these steps in detail.

We will use following convention throughout this section that our task is to transliterate from *Roman* (the 'source' script) to *Devanagari* (the 'target' script). We will use r to denote a word in Roman script and r is sequence of characters $r_1 r_2 ... r_m$ where m is length of a word in Roman script. We will denote a word Devanagari script by d and it is composed of sequence of characters $d_1 d_2 ... d_l$ where l is length of a word in Devanagari script.

We have a training data of n pair of words with each pair represented as (r^k, d^k) for $k = 1, 2..., n$. Here, r^k and d^k denotes the k^{th} word in Roman and Devanagari script respectively. Each k^{th} word in Roman script will be sequence of characters $r_1^k r_2^k ... r_{m_k}^k$ where m_k is length of k^{th} word in Roman script.Each k^{th} word in Devanagari script will be sequence of characters $d_1^k d_2^k ... d_{l_k}^k$ where l_k is length of k^{th} word in Devanagari script.

3.2.1 Language model for Marathi

Language model (LM) is used to ensure the fluent output. Hence, it is built using target language (in our case, Marathi) corpus. The language model assigns a probability $p(d)$ to a word $d = d_1 d_2 ... d_l$ in Devanagari script. We will use character level n-gram models as LMs.

3.2.2 Transliteration model: From Roman to Devanagari script

A transliteration model assigns a probability $p(r|d)$ to any pair of Roman/Devanagari words. $p(r|d)$ indicates how likely we are to see r as a transliteration of d. We use training data to estimate different parameters of the transliteration model. This model makes two choices: First, a choice of length m for a word in Roman script and second, a choice of $r_1 r_2 ... r_m$ which will form r.

To transliterate from Roman to Devanagari, we need to calculate $p(d|r)$. Using Bayes theorem,

$$p(d|r) = \frac{p(d) * p(r|d)}{p(r)} \tag{3.1}$$

The denominator of right hand side is effectively constant. Thus, we only focus on numerator. As explained in Section 3.2.1, we can create language model using Marathi corpus.

Now, we concentrate on the definition of transliteration model $p(r|d)$ and estimation of various parameters based on training data. Instead of directly defining $p(r_1 r_2 ... r_m | d_1 d_2 ... d_l)$, idea is to define alignment variables $a_1 a_2 ... a_m$ i.e. for each character in Roman script word, which will take values in $0, 1, ..., l$. Thus, for every Roman character will have alignment with some Devanagari character. With help of alignment variables, we define a transliteration model $p(r_1 r_2 ... r_m, a_1 a_2 ... a_m | d_1 d_2 ... d_l, m)$ over all words in Roman script.

To get $p(r_1 r_2 ... r_m | d_1 d_2 ... d_l)$, we sum above model over all alignment variables.

$$p(r_1 r_2 ... r_m | d_1 d_2 ... d_l) = \sum_{a_1=0}^{l} \sum_{a_2=0}^{l} ... \sum_{a_m=0}^{l} p(r_1 r_2 ... r_m, a_1 a_2 ... a_m | d_1 d_2 ... d_l, m) \tag{3.2}$$

We now describe different alignment models.

3.2.2.1 Alignment Models

Consider a word in Roman script $r = r_1...r_m$ paired with a word in Devanagari script $d = d_1...d_l$, the alignment variables $a_1...a_m$ - one variable corresponding to one Roman character in a word - where each alignment variable will take values from $0, 1...l$. (Note that 'zero' corresponds to NULL character). These variable will specify which Devanagari characters in a word is aligned with a Roman character in a word. We now explain these models briefly.

3.2.2.1.1 IBM Model - 1

Model 1 consists of finite set D of Devanagari characters, a set F of Roman characters, and integers M and L specifying the maximum length of Roman and Devanagari words respectively. The parameters of the model are as follows:

- $t(r_i|d_j)$ for any $r_i \in R$, $d_j \in D \cup \{NULL\}$. It can be interpreted as the conditional probability of generating a Roman character r_i from a Devanagari character d_j.
- $q(j|i, l, m)$ for any $l \in 0, 1..., L$, $m \in 0, 1..., M$, $i \in 0, 1..., m$, $j \in 0, 1..., l$. It can be interpreted as the probability of alignment variable a_i taking value j conditioned on length $l\&m$ of Devanagari and Roman words respectively.

In Model 1, we assume that, for all i, j, l, m

$$q(j|i, l, m) = \frac{1}{(l + 1)} \tag{3.3}$$

Thus we have uniform distribution over all l+1 Devanagari characters in a word. (Due to $NULL$, we have an extra character in Devanagari word. Thus denominator is $(l + 1)$
Given these definitions, for any Devanagari word $d = d_1...d_l$ where each $d_j \in D$, for each length m, we define the conditional distribution over Roman words $r = r_1...r_m$ and alignments $a_1...a_m$ as:

$$p(r_1...r_m, a_1...a_m|d_1...d_l, m) = \prod_{i=1}^{m} \frac{1}{(l + 1)} * t(r_i|d_{a_i}) \tag{3.4}$$

To estimate parameter t, Expectation Maximization (EM) algorithm is used. This always converges to global optimum of log-likelihood function. Thus, irrespective of initial value, the EM algorithm will converge to same value.

3.2.2.1.2 IBM Model - 2

For Model 2, we have same set of parameters as defined in section 3.2.2.1.1. Now we define the conditional

distribution for Model 2.

$$p(r_1...r_m, a_1...a_m|d_1...d_l, m) = \prod_{i=1}^{m} q\left(a_i|i, l, m\right) * t\left(r_i|d_{a_i}\right) \tag{3.5}$$

If we look at the equation 3.5, it can be observed that IBM Model 1 is a special case of IBM Model 2 where:

$$q(j|i, l, m) = \frac{1}{(l+1)} \tag{3.6}$$

For estimation of parameters $t\&q$, EM algorithm is used. However, the algorithm may converge to different estimates depending upon the initial parameter values. This happens because algorithm may converge to different local optima depending upon the staring point. Thus, care should be taken while initializing the parameters. With the help of Model 1, we can adopt following procedure to initialize parameters $t\&q$.

1. For Model 1, estimate parameter t, by EM algorithm
2. Initialize t with values obtained in step 1 & q with random values. Estimate parameters with EM algorithm.

3.2.2.1.3 IBM Model - 3

Model 3 model adds the *fertility* to previous models. We define *fertility* as follows:

Fertility[14] : For a given alignment a_i, *fertility* (Φ_j) of a target (Devanagari) character d_j is number of source (Roman) words aligned to it . i.e.

$$\Phi_j(a_i) = \sum_{i:a_i=j} 1 \tag{3.7}$$

Given the definition of *fertility* and parameters defined in section 3.2.2.1.1, we now define the conditional distribution for Model 3.

$$p(r_1...r_m, a_1...a_m|d_1...d_l, m) = p(\Phi_0|m).\prod_{i=1}^{m}\left[q\left(a_i|i, l, m\right).t\left(r_i|d_{a_i}\right)\right].\prod_{j=1}^{l}\left[\Phi_i!p(\Phi_i|d_i)\right] \tag{3.8}$$

where $p(\Phi_0|m)$ = probability of fertility of NULL word

In his workbook on Statistical Machine Translation[8], *Kevin Knight* has explained how to calculate different parameters involved in the above formula. *Och & Ney* has also introduced a modification in [13].

Model 3 has two practical problems:

- EM algorithm does not converge to global optimum
- EM algorithm needs to iterate over all possible alignments for every sentence pair. This is very expensive. This issue can be solved if we iterate over fewer number of "good looking" alignments.

To solve both the issues, we find parameter t using Model 1 and initialize the EM algorithm for Model 3 with this parameter.

3.2.2.1.4 Hidden Markov Model (HMM)

All IBM Models, we have described so far, have alignment variables dependent only on one parameter - Devanagari character. In HMM Model, we will consider alignment variables which will be dependent on previous alignment variable. The reason to do so is that the words in natural languages are not distributed arbitrarily but they tend to form a cluster. This clustering might lead to alignments preserving their neighborhood as explained in [15].
HMM alignment model has Devanagari characters as states and these states will emit Roman characters according to probability distribution $t(r_j|d_i)$. This model has two components in it as explained in [2].

1. Alignment component :- It models transition between two states.
2. Translation component :- It models emission of Roman characters from states.

In this model, the alignment variable depends only on previous alignment. For this changed assumption, we change the definition of q which was earlier defined in section 3.2.2.1.1

- $q(a_i|a_{i-1}, l)$. This can be interpreted as probability of emitting from state a_i given last was emitted from a_{i-1} where $1 \leq a_i, a_{i-1} \leq l$.

With these new definitions, we now define conditional distribution for HMM alignment model.

$$p(r_1...r_m, a_1...a_m|d_1...d_l, m) = \prod_{i=1}^{m} q(a_i|a_{i-1}, l) * t(r_i|d_{a_i}) \tag{3.9}$$

The parameters $t\&q$ are estimated separately. Parameter t can be estimated using the methods which we used for IBM Models. To estimate parameter q, we can use jump distance as explained in [2].

3.2.2.2 Testing the transliteration model

To test the transliteration model for a new Roman word, we use models built in 3.2.1 and 3.2.2. We define a set P, which contains words in Devanagari. For a new Roman word r, the output of transliteration system is:

$$d^* = \arg\max_{d \in P} [p(d) * p(r|d)] \tag{3.10}$$

Chapter 4

Experimental Setup

4.1 Data

4.1.1 Data Creation

This section describes how the data for Marathi language was created. As explained in section 2.1, under *Scenario* mode different topics were given to users and they submitted their transliterated content. The contents of each of the document was then back transliterated to Devanagari script manually. While back transliterating code mixing was handled i.e. whenever a word from English dictionary occurred with its correct spelling in transliterated content, the back transliterated word was appended with '*'. The sentences then were arranged with a Roman script sentence followed by Devanagari script sentence. For example:

`he;chitra;pahun;khup;samadhan;vatale`

हे ; चित्र ; पाहून ; खूप ; समाधान ; वाटले

Other than *Scenario* mode, transliterated content was collected from Facebook confession pages, blogs, song lyrics pages etc. This data was then cleaned by removing words like 'sooooooooo','hmmmmmmmm' etc. Cleaned data was, then, back transliterated to Devanagari Script manually and arranged as mentioned above.

4.1.2 Preprocessing

4.1.2.1 Preprocessing for Query Word Labeling

For query word labeling subtask, the pair of sentences were taken and parsed to get tokens. Depending upon, whether the Devanagari word was appended with '*' or not, word was labelled as 'E' (for English) and 'M' (for Marathi). To bring these labelled tokens into the training data format of MALLET, for each token character level unigram, bigram, trigram,four-gram & five-gram were found. Each token with its features occupied a line and it was considered as one of the features. It was arranged in following manner:

[Word] [Label] [Feature_1] [Feature_2] ...

The example of a processed token is as follows:

```
vakya M v a k y a va ak ky ya vak aky kya vaky akya vakya
media E m e d i a me ed di ia med edi dia medi edia media
```

10-fold cross validation was used to train and test different classifiers.

4.1.2.2 Preprocessing for Backward Transliteration

For backward transliteration, a Language Model (LM) for target language (in our case, Marathi) was created with the Marathi news corpus which was available in FIRE 2010 track[1]. News data from 2004 and 2005 was taken to create a corpus to create LM. Each word occupied a single line with a space in between the characters. All occurrences of a word were preserved and words were sorted alphabetically. For training and tuning the transliteration system, separate files were created for Devanagari & Roman words. Like corpus, each word occupied single line with a space in between the characters.
Out of 5362 word pairs that we had collected, 5000 words were chosen as training data and 362 words were chosen as development set. 10-fold cross validation was used to train and test the transliteration model.

4.1.2.3 Preprocessing for Ad hoc Retrieval

For this task, we used the Marathi news corpus which mentioned in section 4.1.2.2 as collection of document on which retrieval would be carried out. Each document was processed to take out document ID and text.

4.2 Tools

4.2.1 Tools for Query Word Labeling

For query word labeling subtask, we need to identify language of each word in query. We already know that the training data contains only two languages (classes).
For identification, we used classifiers in MALLET tool. It included Maximum Entropy, Naive Bayes, Decision Tree and Winnow classifiers.
We will also use WordNet[12][3] to check the presence of English word.

4.2.2 Tools for Backward transliteration

For backward transliteration, Moses[2] tool was used. Though, it is primarily used in Statistical Machine Translation (SMT), it can be used as transliteration system where each character is considered as word and will be compared to the target words to create a transliteration model. With the help of LM and alignments, Moses performs backward transliteration.

[1] http://fire.irsi.res.in/fire/data
[2] http://www.statmt.org/moses/index.php?n=Main.HomePage

For LM creation, IRSTLM[3] tool was used. This creates n-gram Marathi LM which is then used to obtain fluent transliterations.

For alignment, GIZA++[4] tool was used. It contains various IBM Models and HMM model for alignment.

4.2.3 Tools for Ad hoc Retrieval

For ad hoc retrieval, we used Apache Lucene[5] for indexing and searching through the collection.

4.3 Metrics

We will use set of metrics for different subtasks.

- For Query Word Labeling, we will use the mean accuracy over 10 folds for each classifier. Also, we will calculate the accuracy for Marathi and English classes separately based on confusion matrix.
- For Back Transliteration, we will use Moses BLEU metric. It is defined as follows:

$$BLEU = BP.exp\left(\frac{1}{4}\sum_{i=0}^{4}\log p_i\right) \tag{4.1}$$

  where

  p_i = i-gram precision for i = 1, 2, 3, 4

  c = candidate length and r = reference length

  Brevity Penalty (BP) = $\begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases}$

- For ad hoc retrieval, we will use Mean average precision (MAP), Mean reciprocal rank (MRR), P@30 and *bpref*

[3] http://sourceforge.net/projects/irstlm/
[4] https://github.com/moses-smt/giza-pp
[5] https://lucene.apache.org/

Chapter 5

Experiments

5.1 Experiments for Query Word Labelling

As we have explained in previous chapter, we will use MALLET tool to classify the query words. MALLET has two versions, namely, *command line* and *Java API*. We chose to do our experiments with JAVA API version.
It involved following steps

1. We imported the data from input file which stored training instances in MALLET compatible format
2. We divided data into 10-folds, performed training using 9 folds and tested the various classifiers with 1 fold
3. After each iteration, we performed a search through WordNet to check whether a word belongs to English or not

5.2 Experiments for Backward transliteration

In the previous chapter we explained that backward transliteration will be done using Moses, GIZA++ and IRSTLM.
The tools are installed in following directories:

- Moses : `~/Dissertation/mosesdecoder`
- GIZA++ : `~/Dissertation/mosesdecoder/tools`
- IRSTLM : `~/irstlm`

5.2.1 N-gram Language Model (LM) creation

We performed experiments to create language models with different values of N in order to get the higher fluency in the output. Here are the steps.

1. We added start & end symbols for each word and trimmed words with length more than 80 characters.

   ```
   $ irstlm/bin/add-start-end.sh < corpus.mh > corpus.sb.mh
   ```

2. We built an N-gram model, which is stored in intermediate ARPA format. We can change the value of N-grams using option *-n*

   ```
   $ irstlm/bin/build-lm.sh -i corpus.sb.mh -t /tmp -n 4 -p -s improved-kneser-ney -o
   marathi.4.lm
   ```

3. We read the LM in intermediate ARPA format and produced compiled representation which is quick to read and process for the toolkit

   ```
   $ irstlm/bin/compile-lm --text=yes marathi.4.lm.gz marathi.arpa.4.mh
   ```

4. We binarised the compiled vesrion to create binary LM. This is quick to load.

   ```
   $ mosesdecoder/bin/build_binary -i marathi.arpa.4.mh marathi.blm.4.mh
   ```

5.2.2 Training and Tuning the translation system

We trained the translation system with with different alignment models for training data including IBM Models and HMM. Following steps were executed in order to train the system

1. We trained transliteration system based on LM created in section 5.2.1. We can change the LM by *-lm* option, however, we must give the 'n' value which was used to create the LM. The directory which contains GIZA++ toolkit is given by option *-external-bin-dir*. We can select alignment models in GIZA++ with option *–giza-option* and number of iterations after each of the model numbers starting from m1 to m6 & mh (HMM). For choosing HMM, *–hmm-align* option is used.

   ```
   $ nohup nice train-model.perl -root-dir train -corpus train_en_mr -f en -e mh -alignment
   grow-diag-final-and -reordering msd-bidirectional-fe -lm 0:4:marathi.blm.4.mh:8\
   -external-bin-dir /mosesdecoder/tools --hmm-align --giza-option m1=0,m2=0,mh=20,m3=0,m4=0
   >& training.out
   ```

2. We tuned the transliteration system with a small set of word pairs. In tuning process, the tuning dataset is decoded, generating n-best list and weights are updated. With these new weights the tuning set is again decoded. This iterative process continues till convergence under some criterion is achieved.

   ```
   $ nohup nice mert-moses.pl dev_en_mr.en dev_en_mr.mh bin/moses moses.ini --mertdir
   mosesdecoder/bin/ &> mert.out
   ```

5.3 Experiments for Ad hoc Retrieval

We preporcessed the data as explained in subsection 4.1.2.3. With the help of Apache Lucene, we first analyze the data using `WhitespaceAnalyzer` and index it on two fields namely *ID* and *body*. We then search with the queries originally in Devanagari and transliterated to Devanagari. We formulated 50 queries from FIRE 2008 and tested our approach for retrieval with them.

Chapter 6

Conclusions

6.1 Query Word Labeling

We now present result of language identification with various classifiers.

Classifier	**O(%)**	**OR(%)**	**MR(%)**	**ER(%)**
NaiveBayes	96.02	98.41	98.59	95.70
MaxEnt	96.23	98.89	99.25	93.83
DecisionTree	95.44	98.61	98.91	94.39
Winnow	93.49	96.54	96.80	92.91
O:Overall(Learning), OR: Overall(Learning+Rule based) MR:Marathi(Learning+Rule), ER: English(Learning+Rule)				

Table 6.1: Mean Accuracy : 10-fold cross validation

The second column in above table gives us idea about the overall result of 10-fold cross validation with only learning algorithm. Subsequent columns include results which are combination of learning algorithm as well as rule based classification. Significant changes in numbers between percentages of second and third column can be attributed to the skewed dataset which has large number of Marathi words compared to English words.

6.2 Backward Transliteration

We now present result of backward transliteration. We have used BLEU metric which is available in Moses. The precisions mentioned in the results in character level.
The following table shows the results regarding the N-gram models for different values of N. To get this result, we have default settings for alignment model iterations. The default number of iterations for HMM & IBM Model-1 are 5 and for IBM Model-3 & Model-4 are 3.

N	p_1	p_2	p_3	p_4	BP	BLEU
3	90.5	78.8	71.5	64.9	0.977	74.15
4	93.1	85.4	80.8	76.5	0.997	83.45
5	94.7	89.0	85.7	82.8	0.992	87.22
6	95.2	90.1	87.2	84.7	0.990	88.33
7	95.3	90.3	87.4	85.0	0.990	88.54
8	95.2	90.0	87.1	84.6	0.990	88.30
9	94.9	89.6	86.6	83.9	0.992	87.93
10	95.1	89.8	86.9	84.3	0.991	88.10
BP:Brevity Penalty, p_n: n-gram precision & n=1,2,3,4						

Table 6.2: Results for different N-gram Models

As we can see from the above table, 7-gram language model(LM) performs better compared to all other N-gram models in almost every parameter.

Therefore, to compare different alignment models, we will use 7-gram LM. For different alignment models, we have chosen 5 iterations to maintain consistency.

Model	p_1	p_2	p_3	p_4	BP	BLEU
IBM Model-1	94.6	89.5	86.7	84.5	0.988	87.68
IBM Model-2	94.7	89.6	86.8	84.5	0.992	88.09
IBM Model-3	95.2	90.1	87.3	84.9	0.987	88.15
IBM Model-4	95.3	90.1	87.3	85.1	0.988	88.34
HMM	95.3	90.4	87.6	85.3	0.988	88.52
BP:Brevity Penalty, p_n: n-gram precision & n=1,2,3,4						

Table 6.3: Result for different alignment Models

As we can see from the table above, for a 7-gram language model(LM), HMM performs better compared to all other N-gram models in every parameter.

6.3 Ad hoc retrieval

In this section, we present our results regarding the search with the back transliterated queries. We compare this result with original Marathi queries. We have 50 queries and 1095 documents, in total, which have been judged for relevance. For each query, we try to retrieve 1000 documents. However, not every query could retrieve 1000 documents. Following table shows results for original and back transliterated queries.

From the table below, we can come to the conclusion that the transliterated queries do not deteriorate the performance of the retrieval system by large amount. If we had Marathi songs data in both the scripts, we would have searched for queries which that are in both the scripts. We hoped that the result of that

search would include more relevant document from Roman script which will add up to the performance. The original queries should have retrieved more relevant documents compared to back transliterated queries but we have exactly 58 more documents which are retrieved and also relevant.

Type	**Ret**	**Rel**	**MAP**	**bpref**	**P@30**	**MRR**
Original	48186	808	0.2035	0.2202	0.1753	0.3984
Trans	48244	866	0.1953	0.2317	0.1867	0.3961

Trans: Backward Transliterated, Ret: # of retrieved documents
Rel: # of relevant documents

Table 6.4: Results of ad hoc retrieval

6.4 Future Work

6.4.1 Query Word Labeling

To enhance the performance of labeling of English words, we must reduce the skewed distribution of Marathi & English words. We must have more English words in our training data. After training with this data, we may not require extra search through the dictionary.

6.4.2 Backward Transliteration

For backward transliteration, we can improve the results if list of word pairs in both scripts is more extensive. Also, the language model(LM) included around 7 million words (with their frequencies). We can include more words which are from different domain to improve the LM.

6.4.3 Ad hoc retrieval

The result of ad hoc retrieval depends on above tasks. If the above tasks perform more accurately, then this will help retrieval system to improve its result.

We have done this subtask with help of news data which is in Devanagari script. We had collected almost 1800 documents for Marathi songs data which were in both scripts. However, we did not perform relevance judgement on these documents. In future, more songs data should be collected from both the scripts, queries should be formed and relevance judgements should be done on those queries.

Future work also should consider the way in which we search with the query in both scripts. User will enter query in Roman script. So after performing first two subtasks, we will get an alternate query in Devanagari. There can be two ways to search. The search can take place with English words in query keeping in Roman script as it is or English word can also be transliterated to Devanagari (in that case, query has all the words in Devanagari script). Also, search should consider combining both queries i.e. query entered by user & query formed by transliteration. It is our hope that most of the related documents to that particular query should be retrieved if we combine both the queries.

Bibliography

[1] Sowmya V. B, Monojit Choudhury, Kalika Bali, Tirthankar Dasgupta, and Anupam Basu. Resource creation for training and testing of transliteration systems for indian languages. In *Proceedings of the Language Resource and Evaluation Conference (LREC) 2010*, pages 2902–2907. European Language Resources Association, 2010.

[2] James Brunning. *Alignment Models and Algorithms for Statistical Machine Translation*. PhD thesis, Cambridge University Engineering Department and Jesus College, 2010.

[3] Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[4] Spandana Gella, Jatin Sharma, and Kalika Bali. Query word labeling and back transliteration for indian languages: Shared task system description. In *FIRE Working Notes*, 2013.

[5] Parth Gupta, Kalika Bali, Rafael E. Banchs, Monojit Choudhury, and Paolo Rosso. Query expansion for mixed-script information retrieval. pages 677–686. ACM Association for Computing Machinery, July 2014.

[6] Sarvnaz Karimi, Falk Scholer, and Andrew Turpin. Machine transliteration survey. *ACM Comput. Surv.*, 43(3):17:1–17:46, April 2011.

[7] Ben King and Steven Abney. Labeling the languages of words in mixed-language documents using weakly supervised methods. In *Proceedings of NAACL-HLT*, pages 1110–1119, 2013.

[8] Kevin Knight. A statistical mt tutorial workbook. 1999.

[9] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *Annual Meeting of the Association for Computational Linguistics (ACL), demonstration session*, 2007.

[10] David Matthews. Machine transliteration of proper names. Masters thesis, School of Informatics , University of Edinburgh, 2007.

[11] Andrew Kachites McCallum. Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu, 2002.

[12] George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, November 1995.

[13] Franz Josef Och and Hermann Ney. A systematic comparison of various statistical alignment models. *Comput. Linguist.*, 29(1):19–51, March 2003.

[14] Thomas Schoenemann. Computing optimal alignments for the ibm-3 translation model. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, CoNLL '10, pages 98–106, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[15] Stephan Vogel, Hermann Ney, and Christoph Tillmann. Hmm-based word alignment in statistical translation. In *Proceedings of the 16th Conference on Computational Linguistics - Volume 2*, COLING '96, pages 836–841, Stroudsburg, PA, USA, 1996. Association for Computational Linguistics.